



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/693,633	10/23/2003	Joseph S. Beda	13768.783.20.1	8889
47973 7590 11/16/2007 WORKMAN NYDEGGER/MICROSOFT 1000 EAGLE GATE TOWER 60 EAST SOUTH TEMPLE SALT LAKE CITY, UT 84111			EXAMINER WOODS, ERIC V	
			ART UNIT 2628	PAPER NUMBER
			MAIL DATE 11/16/2007	DELIVERY MODE PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.

10/693,633

Applicant(s)

BEDA ET AL

Examiner

Eric Woods

Art Unit

2628

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 03 August 2007.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-35 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-35 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date _____.
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____.
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: _____.

DETAILED ACTION

Election/Restrictions

Applicant's previous election without traverse of claims 1-35 is noted. Applicant's amendments to the claims, filed on 8/3/2007 are acknowledged. Such amendment was in response to a holding that the claims previously filed were elected via original presentation. The new claims conform to the originally elected group.

Response to Arguments

The revised version of the amended claims filed on 8/3/2007 significantly changes claim scope.

The grounds of rejection against claims 1-35 do not stand withdrawn.

The argued limitation of the type converter, object model language, presenter system, etc, were not previously presented, and clearly Lewallen (via incorporating the SVG specification) teaches the markup language, as well as the parser/translator (see Figures 1-2, with bridge element 4, which takes in various types of markup and code, serving as the parser/translator and mapping them to objects, as is further explained in the rejection).

As set forth in the grounds of rejection below, the SVG markup language clearly teaches the shape class, the image class, the canvas class, and element class, each of which contains the property specification within that element. Also, as further set forth below, the SVG language clearly suggests the utilization of a video element.

The bridge 4 utilized in Lewallen clearly maps (Figures 2 and 4) the incoming markup / code to different objects in the UI hierarchy, the SVG hierarchy (Figure 4), or

Art Unit: 2628

to different native objects, as shown in Figure 2, where these are clearly visual API objects (as Java is utilized, as explained in the ground of rejection below), and the objects within the SVG hierarchy are also 'visual.'

The presenter system lies within the system of Steele, that takes in the SVG tree, converts it to binary format (BF), and thusly generates the trees as set forth in Figures 5, 6, 7, and 9, especially as the translator is shown in Figures 6 and 7.

Finally, as the rejection below discusses, the SVG markup language is translated into a hierarchy / tree (see Figure 4, Lewallen), which is then translated into visual API calls (Java, Figures 2 and 4), which are then translated to the graphics objects that are then shown on the screen, where the graphics objects exist within the scene graph of Eleftheriadis after they are translated.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

The factual inquiries set forth in *Graham v. John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 1-13 and 18-30 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Lewallen (US 6,675,230 B1) in view of the SVG specification (incorporated by reference by Lewallen (since it utilizes SVG as the DOM implementation), version 1.1, see previous citation for date)(hereafter referred to as 'SVG1'), Eleftheriadis (US 6,092,107 A), and Steele (US PGPub 2004/0110490 A1).

As to claim 1,

In a computing environment, a computer-implemented method for composing vector graphics, the method comprising: (Lewallen 3:35-4:20 utilizes SVG)

-An application programming interface receiving a plurality of function calls for composing vector graphics, (Lewallen – Scalar Vector Graphics (SVG), which is a subset of eXtensible Markup Language (XML), Figure 1 (mixed) – 11:18-30, 11:45-12:20, utilizes the W3C DOM APIs to receive mixed programs (2a, 2b, 2c) containing function calls – 5:1-25)

-The application programming interface being configured to receive function calls for composing vector graphics, the function calls comprising calls defined by a markup language, the markup language providing a string format (SVG has a "string format"; Lewallen shows that Java and W3C API / native commands have string format as well; Figure 1, elements 2a-2c; see state flow diagram for processing objects in Figure 3a) and an object notation for describing a resource (Lewallen Figures 1-3b, objects are listed (UI, Java, DOM)(3:40-60; 5:10-50; 5:60-65, etc)), and the markup language (Lewallen 3:40-65, 5:10-50 – DOM in general; specifically SVG1 sections 1, 4, etc and SVG section 2.2 'Graphical Objects') comprising an element class (see as

Art Unit: 2628

example SVG1 integral DOM interfaces (section 8.5)), **the element class comprising a shape class** (SVG1 section 9 ('Basic Shapes'), which are 'classes' (see 9.5 exemplary)), **an image class** (SVG1 section 13.5, 13.5.1 'Pattern Content Set' containing pattern.class, described as painting a raster image (3.6.2); SVG1 2.3 'img' element), **a video class** (SVG1 teaches animation; SVG1 19 introduces the 'animate' element class, the ability to specify transforms, etc; SVG deals with video per se (user agent definition (section 1.6) include graphical desktop browsers...) and includes it as a resource element (e.g. *video clip* – link via <a> element, SVG1 17.1. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify SVG to have a video tag, in order to allow it to more easily and natively handle video and multimedia elements (Lewallen 14:29-35)), **and a canvas class** (SVG 'canvas' element (defined 1.6, explained in 7.1)),

-The function calls comprising calls defined within the markup language (SVG clearly provides that structure – see sections 1-3 for structure of element grammar.

Lewallen Figure 1 clearly illustrates this process where Figure 3a shows the process of conversion, see also 6:1-7:55, graphics instructions in SVG are converted via the W3C API to graphic API components / elements, which is the purpose of the bridge 4)

-A parser/translator, the parser/translator configured to: (Lewallen 4:7-20 teaches a parser / translator (11:50-63).)

a) Interpreting graphics instructions teach of string format calls and object notation calls within the markup language (Figures 1, 4 - Lewallen teaches that the Bridge serves as the parser / translator as well; direct code calls are correlated to native

O/S objects (18) via object table 24, where object model code calls are directed to Java objects via object mapping 20 of Java objects, and HTML/XML/SVG based markup is translated via W3C DOM API mappings (8) – 5:48- 7:25. Element 200 in Figure 4 takes DOM object calls and sends them to the UI API creating document object 226, which contains embedded DOM document tree based on SVG – 7:34-8:55 / 14:50-67, where W3C API generates native Java objects)

b) Accessing the type converter, the type converter configured to convert string format calls to visual API objects, and, ((Lewallen Figure 1 clearly illustrates this process where Figure 3a shows the process of conversion, see also 6:1-7:55, graphics instructions in SVG are converted via the W3C API to graphic API components / elements, which is the purpose of the bridge 4. Lewallen Figures 1, 4 – the bridge 4 serves to convert graphics instructions in string format – SVG represents markup (11:17-12:20)-mixed statement programs 2(a, b, c) are translated to the other objects via DOM (7:34-8:55; 14:50-67))

c) Adding elements to an element tree within an element and layout system; and (Lewallen 14:50-67: (in part) – “...For instance, if the Internet Explorer receives an Internet Explorer createElement command to create an SVG element, then the Internet Explorer stub factory would call the SVG stub factory to create the new SVG element and the Java object for this SVG element, which the Internet Explorer stub factory would then include as a node in the DOM tree. If the new element, e.g. SVG element comprised a DOM tree of elements maintained in a separate file, then the next W3C command in the mixed statement program would likely be a command to set

Art Unit: 2628

a SRC (source) attribute for the newly created element..." The alternate version of element 4 in Figure 1 is shown as the bridge in Figure 4, where SVG is taken and transformed to the UI APIs, where the DOM document tree provides the location for the calls (11:50-12:20). Clearly the system of Lewallen *lays out* items (e.g. Figure 4) via the element tree, so it serves as a 'layout system' and clearly since it handles SVG it generates elements and so is an 'element system.')

d) Communicating resource-level data to a visual API; (Lewallen clearly teaches that the objects are matched to their UI counterparts as shown in Figure 1 between the bridge 4 and UI objects 16 → native level objects 18, with mapping tables as (20, 22) in bridge 4, where this data is provided when the object is passed down the chain (as in Figure 4), where the hierarchy of the various objects (226a/228a, 226b/228b)

Lewallen suggests further intermediate levels, in that the DOM tree is intermediate (12:18-60) and the Java developer is free to expose data in any object in the user interface (Figure 4, element 220 collectively, UI object document 228), including DOM trees (228) to java tools, and suggests database access (9:10-25), but fails to expressly teach, while Steele teaches:

-An element and layout system, translating data comprising an element tree into calls to the visual API (Steele Figures 3, 5, and 6 collectively; the SVG compiler in 310

Art Unit: 2628

generates the visual application and translates SVG DOM configurations into BF form)

-The visual API:

a) Accepting input from the parser translator, from the element and layout system, and directly from non-markup programming languages, (Steele clearly teaches that where vector elements 610 and behavior elements 620 exist, where these form scene graph – see Figure 7, which shows the visual elements 610 as a visual graph [0051-0058], see 0051-0053 and 0057-0058 specifically. The other elements are represented as a sequence graph 800. These graphs are matched to each other, as in elements 920 and 930, where the original SVG is shown in Figure 910. Clearly the Sequence Graph portion manipulates the visual object layer, such that manipulating the Visual Graph to change locations does the animation of an object, create new objects, etc. SVG can supply animation commands that discuss where the objects are located and the like with attribute and animate commands. Finally, it is clear that the SVG objects shown in Steele have properties associated with them such as size and/or animation information (as an example, see Figure 9).)

b) Interfacing with a scene graph, (Steele Figures 5-7 and 9)

It would have been obvious to one of ordinary skill in the art at the time the invention was made to convert the UI data of Lewallen, stored in object format as in element 228, into a final format (e.g. BF), as set forth by Steele in [0010,0013,0037], etc, because it

Art Unit: 2628

allows for the transmission of less information over the air and further also minimizes processing the end device.

Lewallen and Steele teach most of the limitations of the above claim, but fail to expressly teach, while Eleftheriadis teaches:

c) Being configured to create scene graph objects, and (BIFS format shown in Figures 2-3)

d) Providing access to a compositing and animation engine; (Eleftheriadis (3:1-30) emphasizes: "The use of application programming interfaces (APIs) has been long recognized in the software industry as a means to achieve standardized operations and functions over a number of different types of computer platforms...In the field of graphics, Virtual Reality Modeling Language (VRML) allows a means of specifying spatial and temporal relationships between objects and description of a scene by use of a scene graph approach.... To enhance features of VRML and to allow programmatic control, DimensionX has released a set of APIs known as Liquid Reality. Recently, Sun Microsystems has announced an early version of Java3D, an API specification which among other things supports representation of synthetic audiovisual objects as scene graphs..." This portion teaches the use of APIs, and the industry-recognized benefits of having a scene graph. Next, the concept of a scene graph is shown as element 225 (5:60-6:10) in Figure 2, with the scene graph API 210 in that Figure as well, and explained in 3:1-30, note Figure 1 with the MPEG app 100 that can directly call

Art Unit: 2628

elements in the underlying BIFS and media decoders and such via scene graph API and decoder API. MPEG-4 uses a scene graph data structure (3:39-45), where the underlying decoded objects are positioned with respect to each other based on the BIFS scene graph (6:35-45)(12:40-50, Figure 5). In the end, clearly Eleftheriadis teaches that the scene graph is populated with objects as manipulated by the API (see 5:64-67).)

-A compositing and animation engine interfacing with the visual API, and the compositing and animation engine managing the composing, animation, and rendering of the scene graph; (Eleftheriadis clearly shows in Figure 2 such an engine, which would interface with the visual API of Lewallen/Steele)

-Causing a change in a graphics display in response to modification of data in the scene graph. (Eleftheriadis teaches clearly that the manipulated objects are displayed – see Figure 2, where the output of the decoded BF is routed through renderer 284 through bus 174 to display 285, where this would clearly constitute a ‘display interface.’)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to have a scene graph layer included in the graphics system of Lewallen and Steele in the manner set forth by Eleftheriadis because Eleftheriadis further teaches APIs for editing said scene graph, and creating, deleting, and modifying objects found therein, and methods for executing such operations, where such a modification would allow both paths in Lewallen Figures 1 and/or 4 access to the

Art Unit: 2628

same underlying object set (which the bridge does), where Eleftheriadis provides the teaching that allowing such mappings is beneficial (6:35-45, 9:20-45, 12:40-63, etc).

Therefore, for at least the above reasons, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Lewallen / Steele such that it further had a scene graph layer.

As to claim 2,

The method of claim 1 wherein causing data in the scene graph to be modified comprises causing initialization of a new instance of a visual class.

Lewallen does not expressly teach but Reference Steele does not expressly teach this limitation, but implicitly teaches that SVG data is decomposed into scene graphs, a.k.a. trees, (see Figure 7), and again – whenever new visual elements enter the scene, new subgroups are instantiated, which *prima facie* (see SVG specification, section 9) are elements that compose visual objects, which therefore are new instances of a visual class as recited above, since a class as recited by applicant is comparable to the basic ‘shapes’ in SVG (applicant’s specification clearly uses it – 23:1-20 where applicant’s invention adopts all classes and shapes from SVG) and thusly meets the recited limitation. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 3,

Art Unit: 2628

The method of claim 2 wherein causing data in the scene graph to be modified comprises invoking code to associate a transform with a visual object in the scene graph.

Lewallen does not expressly teach but Reference Steele teaches this limitation, as he discloses rotations in [0053] and further states that rotations and other transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088] that any visual element or object can modified. Such modifications *prima facie* must associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera [0053]), as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be scaled. (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.) Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 4,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking code to place a drawing visual into the scene graph.

Lewallen does not expressly teach but Reference Steele teaches this limitation, as for example he teaches the insertion of unique identifiers into media streams [0106], and further [0088] that any visual element or object can modified. Such modifications and insertions *prima facie* must associate code with a suitable / desired insertion as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be

Art Unit: 2628

logically inserted. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 5,

The method of claim 4 further comprising, causing drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual.

Lewallen does not expressly teach but Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing context per se is incorporated into the data structures of Steele (see Figure 7). It further would have been obvious to modify the system of Lewallen to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]).). (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.) Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 6,

The method of claim 2 wherein causing data in the scene graph to be modified comprises invoking code to associate brush data with a visual object in the scene graph.

Lewallen does not expressly teach but Reference Steele does teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ...' and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which *prima facie* justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, reference Steele intrinsically teaches this limitation and it would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the X3D and graphics system of Steele with the SVG and graphics system of Kim as set forth above and the SVG standard specification, and because they serve complementary and supplementary purposes in how they handle graphics and animation, particularly with respect to the standards they utilize, and the SVG standard inherently handles these paint limitations. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, reference Steele intrinsically teaches this limitation. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 8,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1 and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima*

Art Unit: 2628

facie and inherently possess paint data as set forth by the SVG specification above. As such, reference Steele intrinsically teaches this limitation. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with an image with further details provided in section 11.7.5 under the 'image-rendering'

Art Unit: 2628

property. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 11,

The method of claim 10 further comprising, receiving markup corresponding to an image effect to apply to the image.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above, and SVG is inherently a markup language, so the rendering portion of Steele would receive such information (the rendering functionality is inherent in SVG – see section 11.7, 14.4, et cetera) Motivation and rationale are incorporated by reference from the rejection to the parent claim..

As to claim 12,

The method of claim 1 further comprising, receiving markup corresponding to pen data that defines an outline of a shape.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The term 'pen data' used by applicant above is comparable or analogous to

Art Unit: 2628

any set of data defining the outline of a shape, including SVG 'path' data. Section 11.3 of the SVG specification sets forth that a user can fill a path that would correspond to the outline of shape with multiple illustrations provided for this under the 'nonzero' and 'even odd' subheadings – see details on paths -- with further details provided in section 11.3 and 11.4 (the individual strokes that create these effects. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 13,

The method of claim 1 wherein the markup corresponds to a shape class comprising at least one of the set containing rectangle, polyline, polygon, path, line and ellipse shapes.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include Shape classes. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 14,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The SVG specification clearly shows in section 9.1 that rectangles are a basic shape, and further that in 9.2 under Example rect02 that such rectangles can have rounded corners, and code is provided that implements such. Also, the 'Rect' class inherently has geometry-related functions as set forth in the beginning to section 9.2. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. As such, reference Steele shows a rectangle 715 in the scene graph in Figure 7 that intrinsically teaches this limitation. Also, said element can be animated under SVG section 19.2. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 15,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted

Art Unit: 2628

into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Further, the SVG specification sets forth path data in section 9.1 as existing and how a 'path' can define a shape or similar. Both meanings are covered herein. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

Also, said element can be animated under SVG section 19.2.

As to claim 16,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a line in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. A line element can be animated under SVG section 19.2, which is obviously geometric. Line elements are set forth in SVG section 9.5, and their geometric functions. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera].

As to claim 17,

Art Unit: 2628

The method of claim 1 wherein the markup is related to hit-testing a visual in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, Steele teaches or implies navigation in [0067, 0085] – that is, navigation using a UI through a two-dimensional view, which is what any display normally shows. Therefore, given that a portable computer could clearly be used, the user would clearly be interacting with the display. As such, hit testing would be required for user interactivity, as could the system of Steele under the same rationale. The SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such information is clearly communicated in markup language – see section 16.2 for event types and elements transmitted in markup, for example. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 18,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to transforming coordinates of a visual in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this.

Art Unit: 2628

position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Clearly, SVG teaches the animation of visual elements, see section 19.2, which *prima facie* involves transforming coordinates of a visual in the scene graph data, and according to Steele [0052-0053] and a tree of elements can also be transformed [Steele 0052]. Steele clearly teaches data modification in [0061] as set forth above. The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera]. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 19,

The method of claim 1 wherein the markup is related to calculating a bounding box of a visual in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Bounding box calculations are taught in section 7.1 and detailed in section 7.11 where they are calculated. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 20,

The method of claim 1 wherein causing data in the scene graph be modified comprises invoking a function via a common interface to a visual object in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, the SVG specification teaches interfaces in section 4.3 – there are common DOM interface as set forth there. If the intended meaning of applicant was that such interfaces were based in hardware or software, fundamentally in reference Steele the user interacts with the browser that would provide a common interface, in that all events generated by such browser would go to an interface – that is, Steele clearly sets forth that his invention has various possible interfaces, depending on the embodiment (e.g. PDA, cell phone, et cetera [0004] and [0007-0008]). Steele clearly teaches data modification in [0061] as set forth above. Lewallen can be regarded as implying a common interface via the use of Java. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 21,

The method of claim 1 further comprising invoking a visual manager to render a tree of at least one visual object to a rendering target.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each

Art Unit: 2628

element has its own element information that controls rendering aspects. Steele clearly teaches data modification in [0061] as set forth above. It is *prima facie* obvious that a 'visual manager' of some form must exist in order to handle formatting issues and precedence in rendering, and Steele teaches such a manager in [0075-0076]. Clearly the rendering information each visual element [Steele 0056-0061] is sufficient such that it is its own 'rendering target' as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 22,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a container object in the scene graph data structure, the contained object configured to contain at least one visual object.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 7 shows a tree derived from an animation is shown – Figure 6 [see Steele 0050 and 0079]. SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use of 'container elements', which are defined as 'An element that can have graphic elements and other container elements as child elements'. Steele clearly teaches data modification in [0061] as set forth above. Clearly, the container object could be the head object of the tree structure shown in Steele Fig. 7. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 23,

Art Unit: 2628

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place image data into the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, visual elements can be covered by or tiled with images as established in SVG section 11.1, where SVG teaches: "...can paint (i.e. fill or stroke) with: ...a pattern (vector or image, possibly tiled) ..." which clearly establishes this, with more detail in section 11.7.5 and 11.12. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 24,

The method of claim 23 wherein causing data in the scene graph to be modified comprises invoking a function to place an image effect object into the scene graph data structure that is associated with the image data.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask for any visual element, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 25,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place data corresponding to text into the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 10.1 of the SVG specification sets forth the use of a 'text' element, and Steele teaches the inclusion of text element 725 in the data tree shown in Fig. 7. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 26,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to provide a drawing context in response to the function call.

Lewallen does not expressly teach but Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has

Art Unit: 2628

those properties associated with it [Steele 0052-0056 and 0059-0061]. SVG is also a subset of XML, and SVG teaches metadata use in section 21.1. Steele clearly teaches data modification in [0061] as set forth above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

It further would have been obvious to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]), and the SVG DOM interfaces in section 4.1-4.4 (SVG specification) clearly provide methods for retrieving drawing information, which would be that context.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

Lewallen does not expressly teach but Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing – see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele

reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele]. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 30,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place animation data into the scene graph data structure.

Lewallen does not expressly teach but Reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 7 shows that each subgroup is shifted a certain amount with x and y coordinates given. Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 19.1 – 19.5 clearly sets forth how each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Therefore, clearly animation data is put into the tree of Fig. 7 Steele, which is clearly a scene graph by every known definition of the term, and a sample SVG XML program is provided in the second page of Fig. 9. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 31,

The method of claim 30 further comprising communicating timeline information corresponding to the animation data to a composition engine.

Lewallen does not expressly teach but Reference Steele clearly establishes in [0051-0054] and Figs. 6 and 9 that animation takes place through the SVG standard.

Section 19.2 of SVG sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly states that animation has a document start and document end, and further in the second to last paragraph that the SVG system indicates the timeline position of document fragments being animated. Further, according to SVG 19.2.2 the animation is by document fragment and object path, which clearly are passed to the system is specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly, the system of Steele performs compositing and rendering [0007, 0011-0012]. Finally, reference SVG teaches that it supports compositing (section 14.2.1). The composition engine would be, for example, the composition engine of Steele, the display interface of Lewallen, or the like. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where the system in steps 835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079] which performs interpolation during the animation process as set forth in the SVG standard, and *prima facie* the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set

Art Unit: 2628

forth in the rejection to claim 31 above. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 33,

The method of claim 32 wherein the composition engine is at a low-level with respect to the scene graph.

Lewallen does not expressly teach but Reference Steele does teach it. Steele in Fig. 15 shows that various programs, including the operating system, are on the flash memory, which in [0136] is specified to contain all the low-level programs of the operating system – graphics is low-level functionality. Since there is no specific graphics unit, all graphics operations and compositing are done by the operating system in the microprocessor, which *prima facie* means that in that embodiment, such graphics are done at a low-level, that is the rendering is done by the operating system at a low level. The scene graph is high-level in that it is embodied in RAM and is held as an abstraction – this is a function of the SVG standard that keeps tree nodes and container nodes as abstract as possible, therefore the embodiment in Fig. 18 must do the same. In any case, low-level composition means that it would be done by hardware that is much faster than software. As such, it would be obvious to modify the device of Steele to use low-level rendering, and in any case Steele has the rendering means set forth in the rejection to claim 32 above, which is *prima facie* entirely hardware. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 34,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

Lewallen does not expressly teach but Reference Steele does teach it. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 6.18 clearly sets forth aural style sheets, that are audio data that each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by definition, SVG animations would be video. Motivation and rationale are incorporated by reference from the rejection to the parent claim.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified comprises changing a mutable value of an object in the scene graph data structure.

Lewallen does not expressly teach but Reference Steele does teach it. Steele teaches in [0014] that one embodiment of his invention changes the visual graph in accordance to changes of the sequence graph, where the visual graph is comparable to the "scene graph" of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate embodiment of this particular limitation.

Art Unit: 2628

Motivation and rationale are incorporated by reference from the rejection to the parent claim.

Claims 28-29 are rejected under 35 USC 103(a) as unpatentable over Lewallen, Eleftheriadis, Steele, and SVG as applied to claim 1, and further in view of Kim et al (US PGPub 2003/0120823 A1)('Kim') and X3D specification, which Kim incorporates by reference.

As to claim 28,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a three-dimensional visual into the scene graph data structure.

Lewallen, Eleftheriadis, Steele, and SVG collectively fail to teach this limitation. The Kim reference clearly teaches this limitation. The Kim reference clearly teaches scene graphs as established in the rejection to claim 1 [0007-0009]. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three-dimensional attributes and properties. This *prima facie* establishes that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D

Art Unit: 2628

specification in software, and, as such, it is software, which *prima facie* uses function calls.

Kim [0007-0008] clearly teaches the use of a scene graph and that X3D requires the construction of such scene graphs from primitives. Kim further teaches that the user can move through a scene [0020, 0026], which clearly establishes that a user is navigating and the scene is constantly being re-rendered, which *prima facie* requires data in the scene graph to be modified. Kim can also use MPEG-4, which clearly involves animation and modification of data in a scene graph, which matches with the Eleftheriadis implementation previously cited.

Kim extols the benefits of three-dimensional graphics in [0001-0007].

Therefore, based on the above teachings, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Lewallen, Eleftheriadis, and Steele to have three-dimensional elements.

As to claim 29,

The method of claim 28 wherein causing data in the scene graph to be modified comprises mapping a two-dimensional surface onto the three dimensional visual.

The Kim reference clearly teaches this limitation, and X3D standard is only cited to clarify certain points. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three-dimensional attributes and properties. This *prima facie* establishes

Art Unit: 2628

that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D specification in software, and, as such, it is software, which *prima facie* uses function calls. Secondly, the X3D standard clearly allows for the incorporation of 2D images onto three-dimensional elements, as stated in X3D 18.2.1 and 18.4.1, particularly 18.4.1, which reads clearly that "browsers may support other image formats ... which may be rendered into a 2D image" and clearly those images can be applied to three-dimensional objects such as those described in 18.3.1 and as defined in 18.2.1-18.2.3. Motivation and rationale are taken from the rejection to claim 28 above.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Eric Woods whose telephone number is 571-272-7775. The examiner can normally be reached on M-F 10am - 6pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Art Unit: 2628

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Eric Woods

11/9/2007


ULKA CHAUHAN
SUPERVISORY PATENT EXAMINER